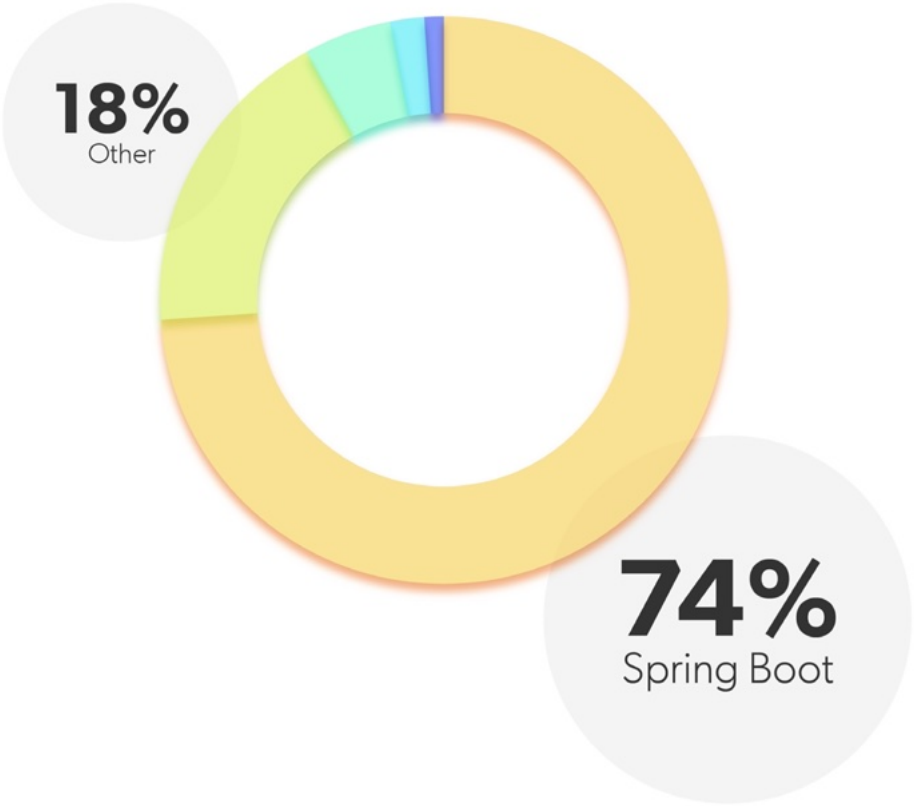
SPRING BOOT 3
SPRING FRAMEWORK 6

MAGNUS LARSSON

CADEC 2023.01.19 & 2023.01.25 |
CALLISTAENTERPRISE.SE

CALLISTA

What Microservice Application Framework are You Using on Your Main Project?



- Spring Boot **74%**
- Other **18%**
- Quarkus **5%**
- Vert.x **2%**
- DropWizard **1%**

AGENDA

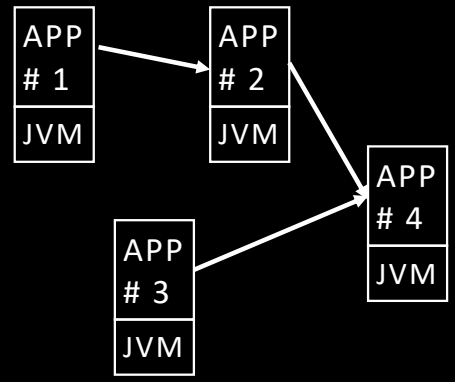
- Overview
- Migration
- Native Compile
- Observability
- Summary

OVERVIEW

- The evolution of **Java**

New Requirements

- Faster startup
  - Shorten warmup
- Less memory
- Scalability
  - Scale to zero
- Observability



Application Servers → **Distributed Systems**

2000

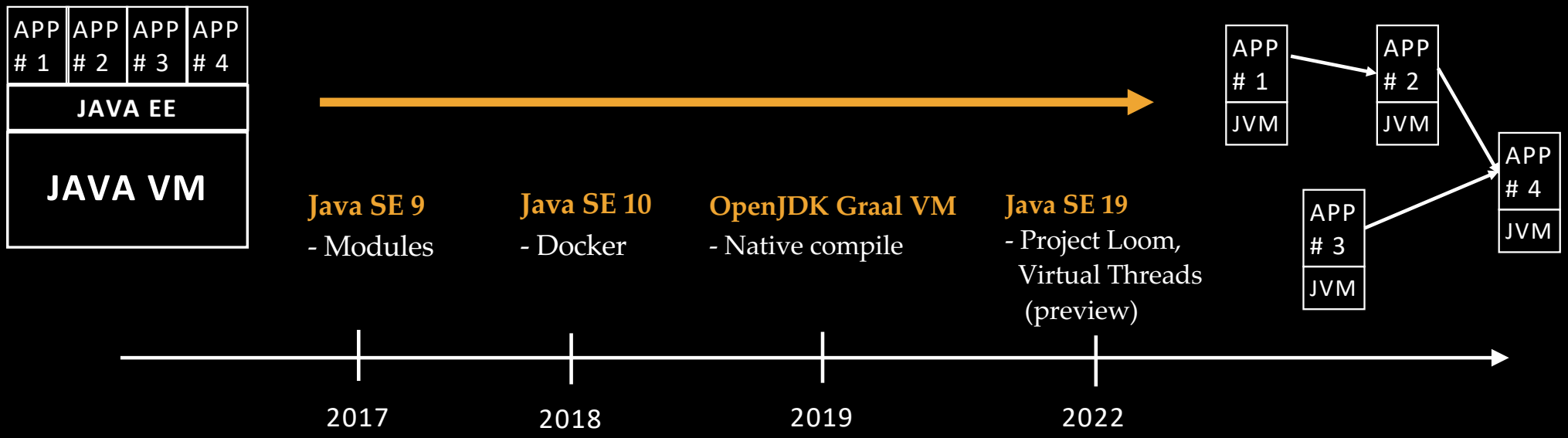
2023

OVERVIEW

- The evolution of **Java**

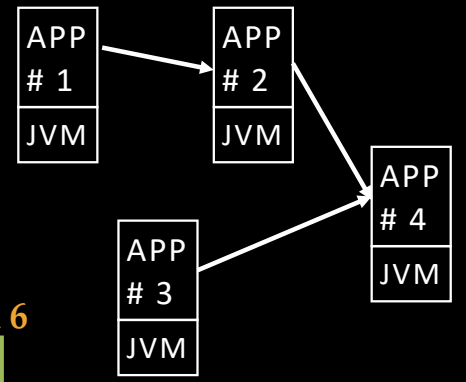
Emerging OpenJDK projects:

- CRaC, Amber, Valhalla, Leyden, and Panama



OVERVIEW

- The evolution of **Spring**



Spring Boot 1
Spring Framework 4

- Convention over Configuration
- Make JAR nor WAR
- Dependency Mgmt, **`starters`** and **`bom`**

Spring Boot 2
Spring Framework 5

- Reactive programming model
- Production features **`actuators`** health, monitoring, metrics

Spring Boot 3
Spring Framework 6

- Native Compile
- Observability

THIS PRESENTATION

- Virtual Threads

NEXT PRESENTATION

2014 2018 2022

AGENDA

- Overview
- **Migration**
- Native Compile
- Observability
- Summary

MIGRATION

- Java 17 baseline
- Jakarta EE: Package rename: `javax` → `jakarta`
  - E.g. JPA: `javax.persistence.*` → `jakarta.persistence.*`
- Deprecated code in 2.x removed

```
tasks.withType(JavaCompile) {
    options.compilerArgs += ['-Xlint:deprecation']
```

- Breaking changes etc
  - Spring Boot 3.0 Migration Guide
  - Spring Security 6.0 Migration Guide
- Importance of end-to-end black-box tests
  - Run them before and after the migration!

AGENDA

- Overview
- Migration
- **Native Compile**
- Observability
- Summary

NATIVE COMPILE

For details on GraalVM Native Image, see this presentation

- Problem definition from Cadec 2021 - GraalVM Native Image

JVM based Micro Services

1. Large memory footprint
2. Long startup time
3. Initial warmup required (JIT)

Makes it expensive for large systems and impossible to scale to zero.

NATIVE COMPILE

- Concerns from Cadec 2021

Develop and Build Findings

- Unable to compile static executable and build from a scratch docker-image
  - Both Go and GraalVM native executables depends on shared C/C++ libraries
  - Googles `gcr.io/distroless/base` is used instead
- Even minor changes breaks the build
  - Spring Boot 2.4.0-RC1 to 2.4.0 release update
  - Graal 20.2 to 20.3 minor update
  - Use of new features from existing 3rd party libraries
  - Adding 3rd party libraries
- Discrepancy between dev and runtime environments
- What's the credibility of unit tests

- Frameworks/libs without native support
- Use and maintain configurations for Reflection, Proxies, Resources and JNI

Road to Enable Native

GraalVM 20.3

1. Upgrade to Spring Boot 2.4
2. Add GraalVM native support. Substrate VM (`svm`)
3. Add Spring native support (`spring-graalvm-native`)
4. Create build script or use maven plugin (build.sh)
5. Declare all Reflections (for DTO beans) and resources
   - Manually or use `native-image-agent` to generate
6. Compile, run and fix remaining stuff (trial and error)
   - Reflection config for Kafka and JSON serializers
   - Resource config for Kafka
   - Substitute Kafka class using Method Handles

Requires Configuration

- Reflections, Dynamic Class Loading
- Dynamic Proxies (JDK)
- Resource Access
- Java Native Interface (JNI)

```
native-image --initial
[total]: 687,593.47 ms
```

NATIVE COMPILE

- Outcome from Cadec 2021

Dear fellow JVM'ers!

"There's no Holy Graal, just loads of hard work and Java."

- Me

- Is it better now?

NATIVE COMPILE

- With Spring Boot 3 and Spring Framework 6
  - Compile Spring Boot applications into standalone executables, called a **native image**
  - Uses GraalVM `native-image` compiler
    - » New build module Spring AOT
    - » Supersedes Spring Native

- Benefits
  - Shorter startup times
  - No warm-up required
  - Less memory required
  - Fit for scaling up and down
    - » Even to zero

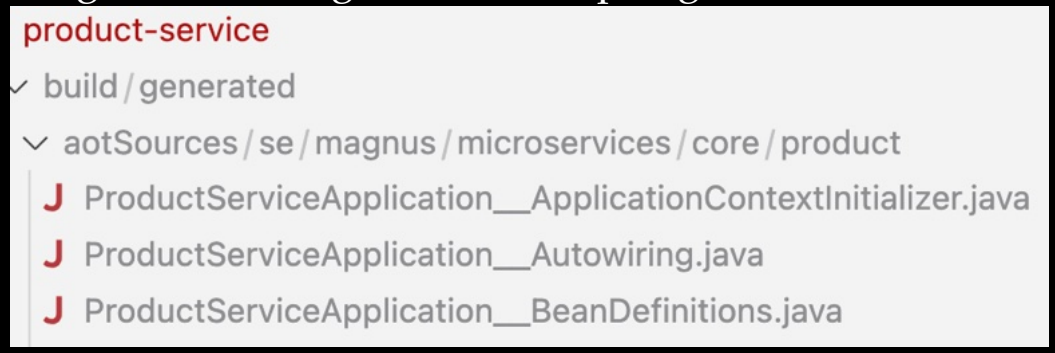
NATIVE COMPILE

- Spring AOT
  - Creates and inspects an `ApplicationContext`

```
public static void main(String[] args) {
  ApplicationContext ctx = SpringApplication.run(ProductServiceApplication.class, args);
```

  - Closed world assumption
    - » Classpath fixed and Spring Beans are defined at build time
    - » Minimize memory footprint
  - Generates start-up code
    - » Creates a static `ApplicationContext`
    - » Programmatic registration of Spring Beans

Replaces the slow reflection based startup



NATIVE COMPILE

- Spring AOT
  - Generates native configuration

- Recall from Cadec 2021:
  - GraalVM `native-image` compiler transforms Java bytecode to an executable image
  - Can't figure out dynamic behavior
    - E.g. use of reflection, dynamic proxies, and local resources
  - Described in a native configuration

NATIVE COMPILE

- Spring AOT
  - Generates native configuration
  - Sample of generated native configuration

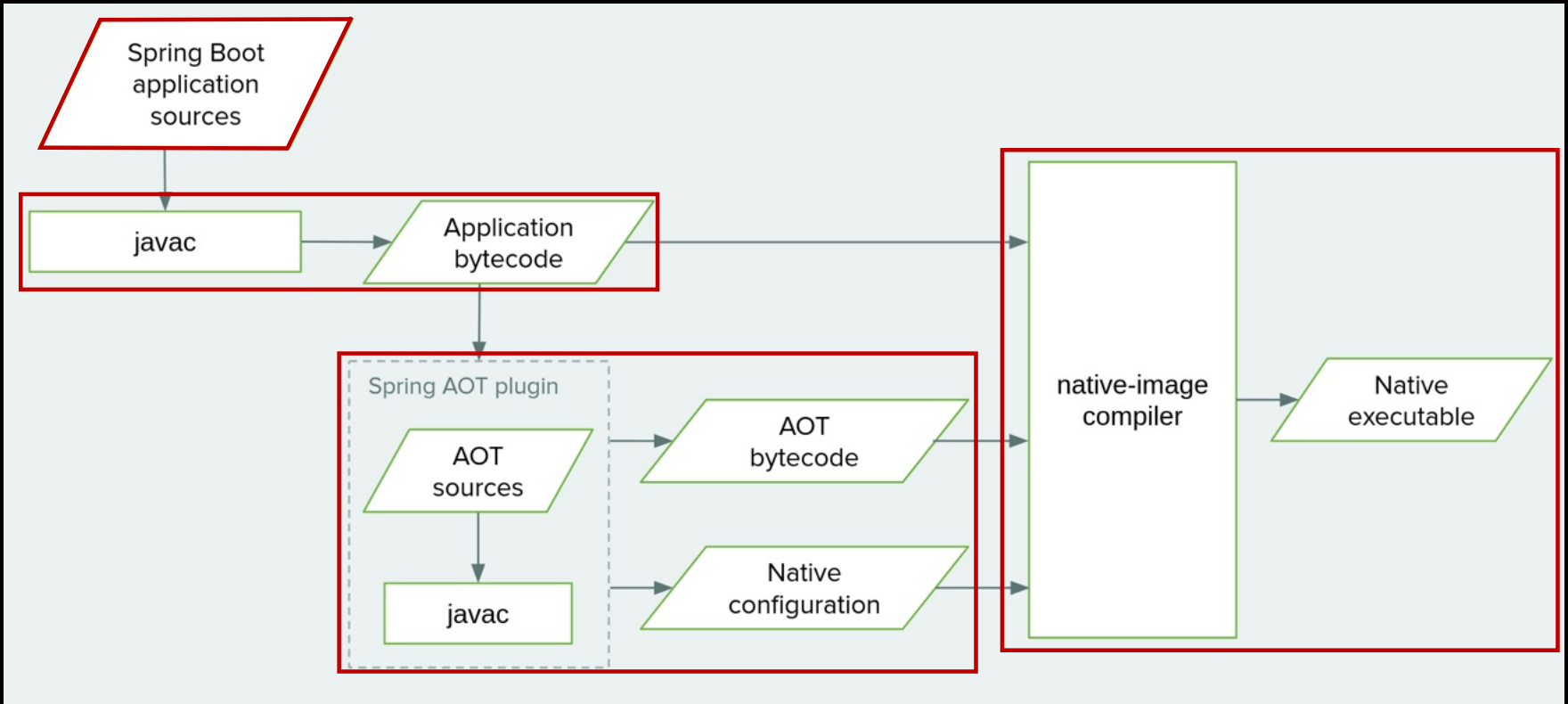
  - If Spring AOT fails, we can add native hints
    - » E.g. JSON mapping with Jackson

```
@RegisterReflectionForBinding({Event.class, Product.class})
```

    - » Sample error message:

```
Error: No serializer found for class se.magnus.util.event.Event
This appears to be a native image, in which case you may need to configure reflection
```

NATIVE COMPILE



https://spring.io/blog/2021/12/09/new-aot-engine-brings-spring-native-to-the-next-level

NATIVE COMPILE

- More on Spring AOT

  - AOT tests
    - » Builds a native image and runs tests inside it

      ```
      ./gradlew nativeTest
      ```

    - » Detects missing Spring Beans and Reflection metadata
    - » Best to run in a CI/CD build pipeline

  - Use AOT start-up code with Java VM (a.k.a AOT mode)
    - » Shorten startup time in Java VM with ≈20%

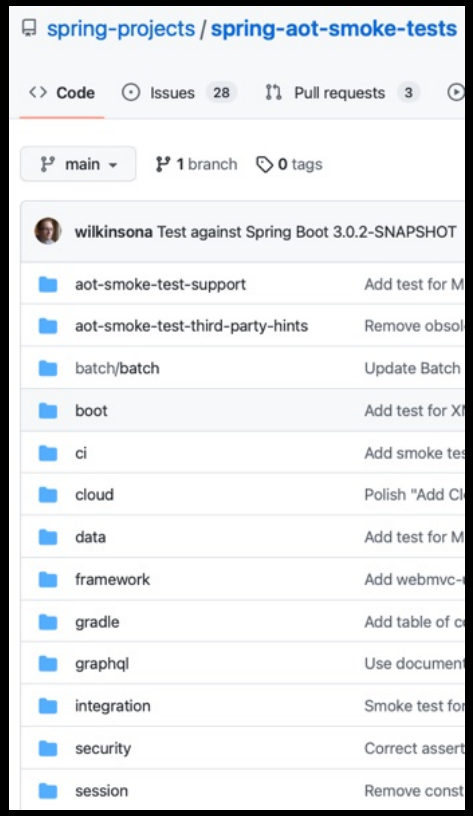
      ```
      java -Dspring.aot.enabled=true -jar app.jar
      ```

    - » Log output

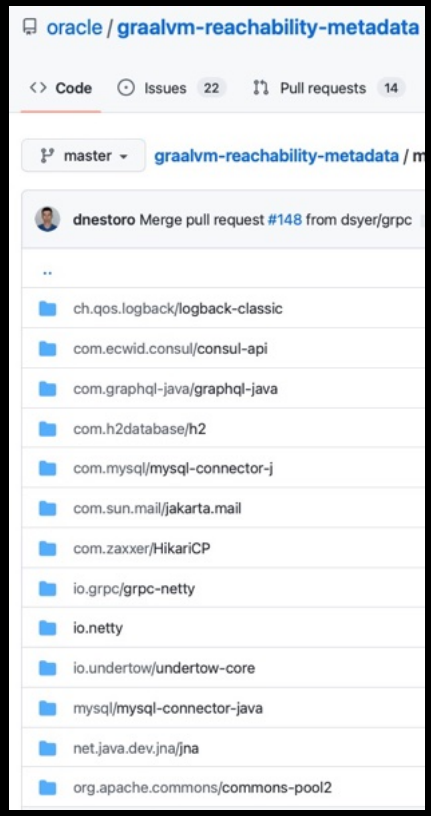
      ```
      ... Starting AOT-processed ProductServiceApplication using Java 17.0.5 ...
      ```

NATIVE COMPILE

- Stability over time
  - Spring ecosystem
    - » Spring AOT smoke tests



  - 3PP libraries
    - » GraalVM reachability metadata



NATIVE COMPILE

- How?
  - Add GraalVM's build plugin:

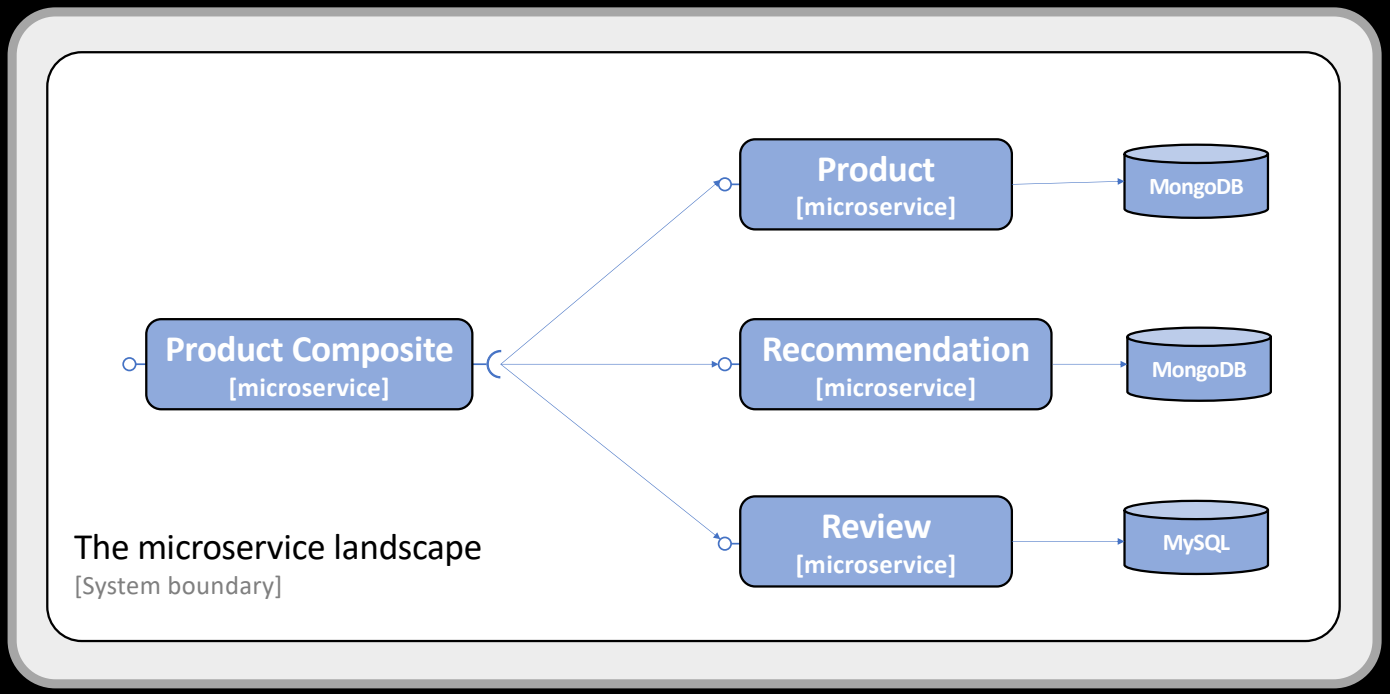
```
plugins {
    id 'org.graalvm.buildtools.native' version '0.9.18'
```

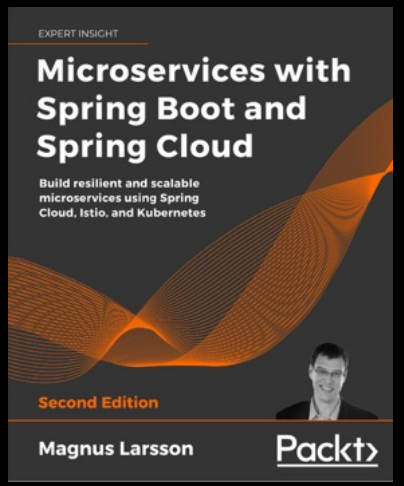
  - Build a native image: **`./gradlew nativeCompile`**
    - » Requires GraalVM JDK and native-image compiler
    - » OS & HW specific, e.g. macOS and ARM64
  - Build Docker image: **`./gradlew bootBuildImage`**
    - » HW specific, e.g. Intel x86_64

- Ongoing work
  - GraalVM - Cross-compilation support
  - Paketo buildpacks - Add support for ARM64
  - Paketo buildpacks - 2023 Roadmap
  - Callista blog post - Docker images on ARM64

TEST: NATIVE COMPILE

- System landscape from the 2ed of my book
  - Migrated to Spring Boot 3





TEST: NATIVE COMPILE TIMES

- Compile times with `./gradlew nativeCompile`
  - Minimal Spring Boot app

```
Finished generating 'demo' in 32,3s.
    [native-image-plugin] Native Image written to: /Users/magnus/
```

  - The Product service from the test landscape

```
Finished generating 'product-service' in 1m 41s.
    [native-image-plugin] Native Image written to: /Users/magnus/
```

  - Not fast enough for a TDD – loop, but sufficient for a CI/CD build pipeline
  - But significantly better than 2021

```
native-image --initial
[total]: 687,593.47 ms
```

TEST: STARTUP TIMES

- Java VM microservices
  - Started ProductServiceApplication in 4.988 seconds
  - Started ProductCompositeServiceApplication in 5.495 seconds
  - Started ReviewServiceApplication in 5.442 seconds
  - Started RecommendationServiceApplication in 4.886 seconds

- Native image microservices
  - Started ProductCompositeServiceApplication in 0.148 seconds
  - Started RecommendationServiceApplication in 0.198 seconds
  - Started ProductServiceApplication in 0.184 seconds
  - Started ReviewServiceApplication in 0.229 seconds

Native image app starts up 25 times faster than JVM app

TEST: MEMORY USAGE AFTER STARTUP

- Java VM microservices
  - review-1 239.2MiB
  - product-composite-1 216.5MiB
  - product-1 212.6MiB
  - recommendation-1 215.5MiB

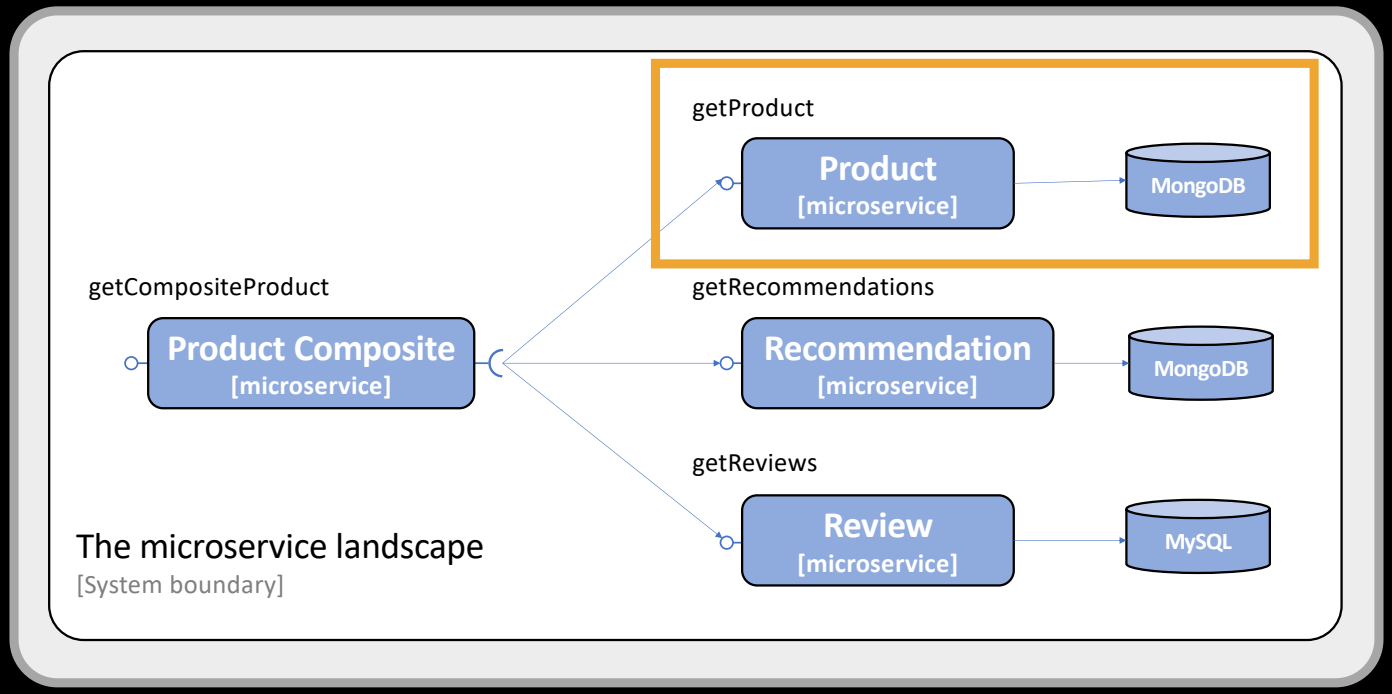
- Native image microservices
  - product-1 78.53MiB
  - recommendation-1 78.55MiB
  - product-composite-1 55.84MiB
  - review-1 70.14MiB

Native image app
requires less memory
to startup.

But what happens over time?

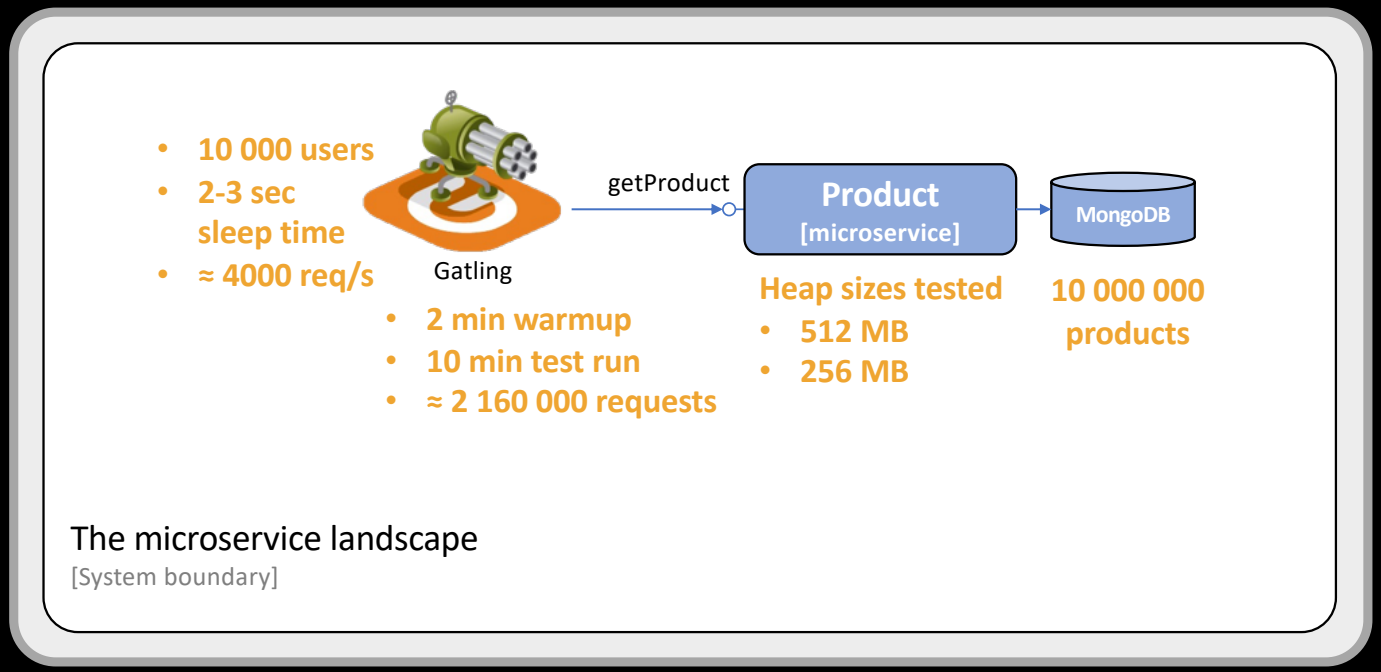
TEST: RESOURCE USAGE OVER TIME

- Test scope



TEST: RESOURCE USAGE OVER TIME

- Test setup



TEST: RESOURCE USAGE OVER TIME

- Test results

Test case	No of calls	RSS mem (MB)	CPU time (min:sec)	Threads	50 % (ms)	95 % (ms)
Native, 256 MB	2 159 056	206	20:54	42	2	11
JVM, 256 MB	2 162 800	235	15:43	54	2	5
Native, 512 MB	2 158 443	220	20:23	42	2	9
JVM, 512 MB	2 163 090	347	15:52	54	2	5

NATIVE COMPILE

- Reiterate the concerns from Cadec 2021

Develop and Build Findings

- Unable to compile static executable and build from a scratch docker-image
  - Both Go and GraalVM native executables depends on shared C/C++ libraries
  - Googles g
- Even mino
  - Spring Boo
  - Graal 20.2
  - Use of new features from existing 3rd party libraries
  - Adding 3rd party libraries
- Discrepancy between dev and runtime environments
- What's the credibility of unit tests

- Experimental Spring Native replaced by Spring AOT
- Tests can run in AOT mode

Road to Enable Native

GraalVM 20.3

1. Upgrade to Spring Boot 2.4
2. Add GraalVM native support. Substrate VM (`svm`)
5. Declare all Reflections (for DTO beans) and resources
   - Manually or use `native-image-agent` to generate
6. Compile, run and fix remaining stuff (trial and error)
   - Reflection config for Kafka and JSON serializers
   - Resource config for Kafka
   - Substitute Kafka class using Method Handles

- Simply add GraalVM's build plugin

- In general, much less of a problem
- When needed, use Spring annotations
- Worst case, use GraalVM's Tracing Agent

- Fram
- Use a
  JNI

s and

- Spring smoke test project
- GraalVM reachability metadata project

- Native compile in a minute or two

NATIVE COMPILE

- Summary
  - Concerns from Cadec 2021 mitigated with Spring Boot 3

  - **Startup**: Native 25 times faster than JVM
  - **Memory**: Native beats JVM
  - **CPU**: JVM Hotspot beats native

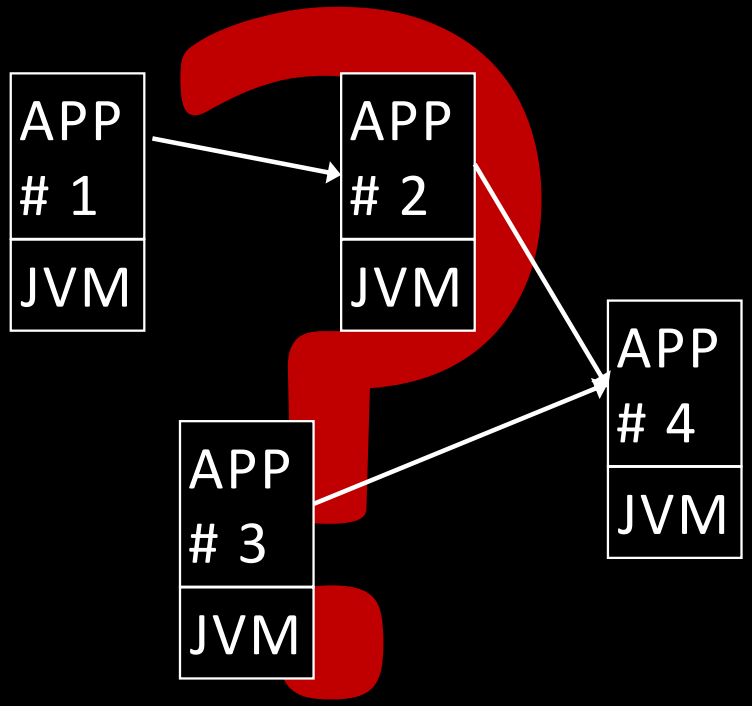
  - Try it out, if start-up time is important!

AGENDA

- Overview
- Migration
- Native Compile
- **Observability**
- Summary

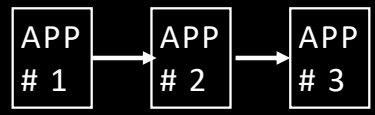
OBSERVABILITY

- Observability = Logging + Tracing + Metrics

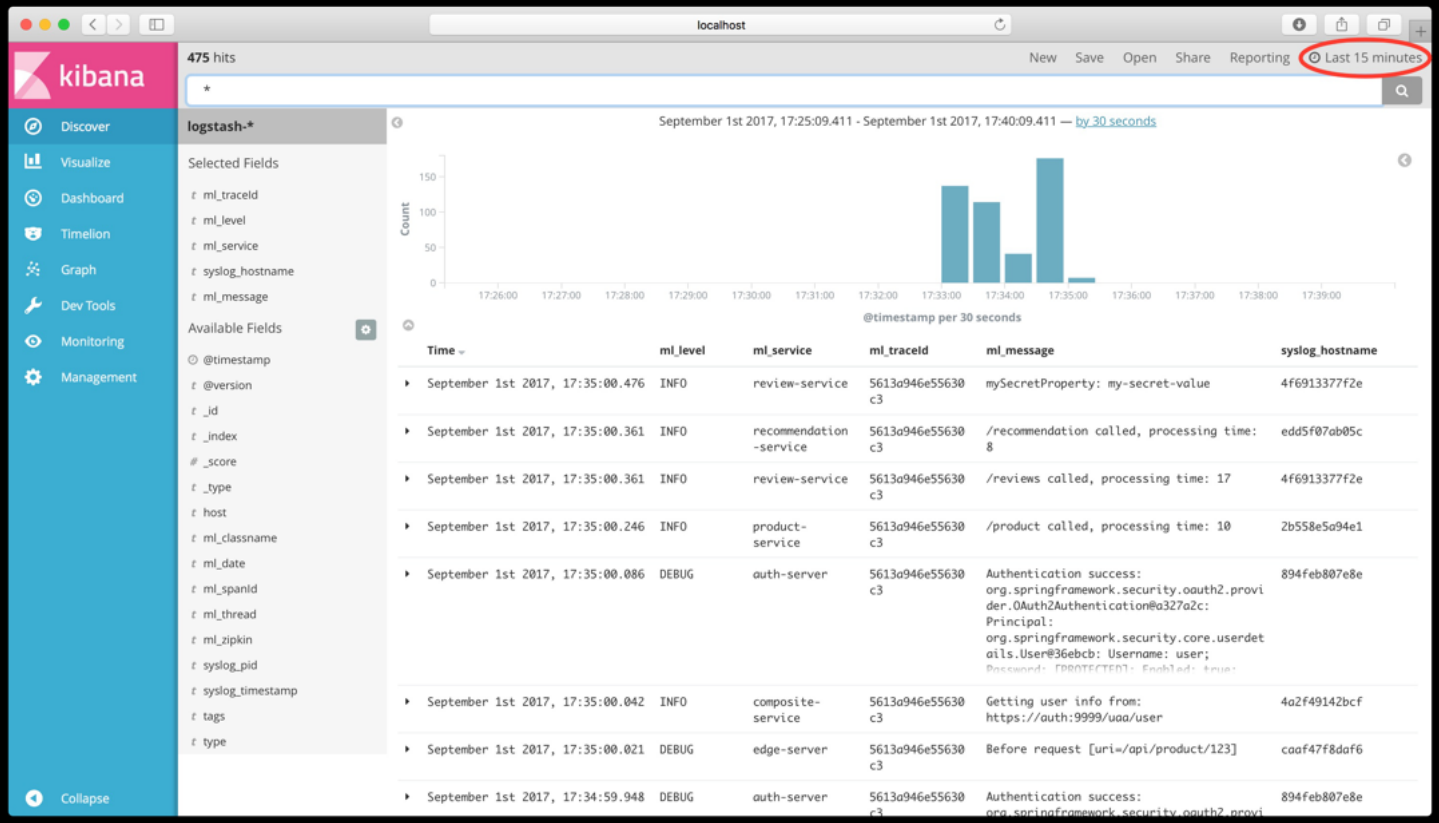


OBSERVABILITY

- Observability = **Logging** + Tracing + Metrics

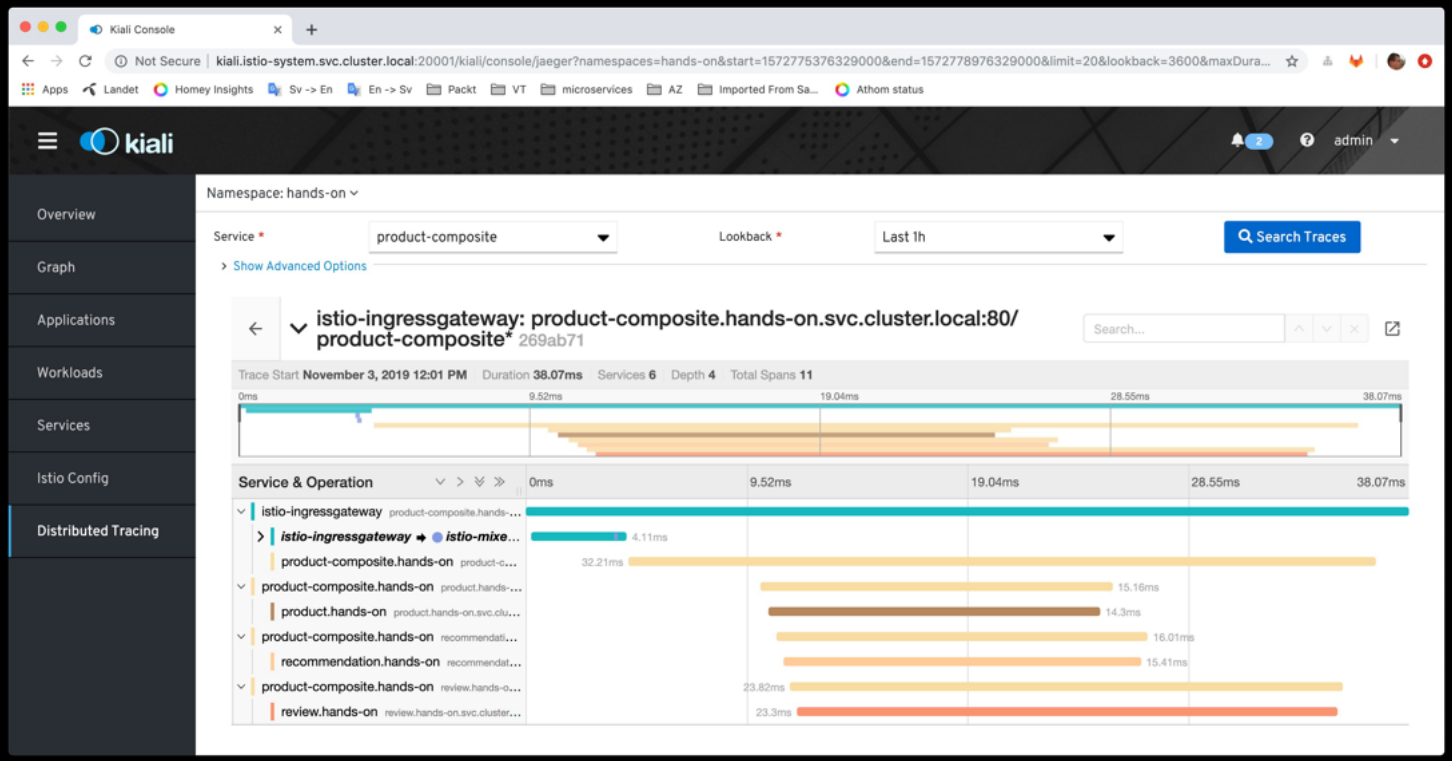


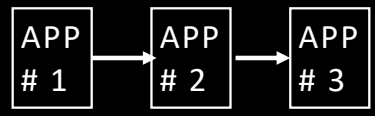
WHAT HAPPENED?



OBSERVABILITY

- Observability = Logging + **Tracing** + Metrics

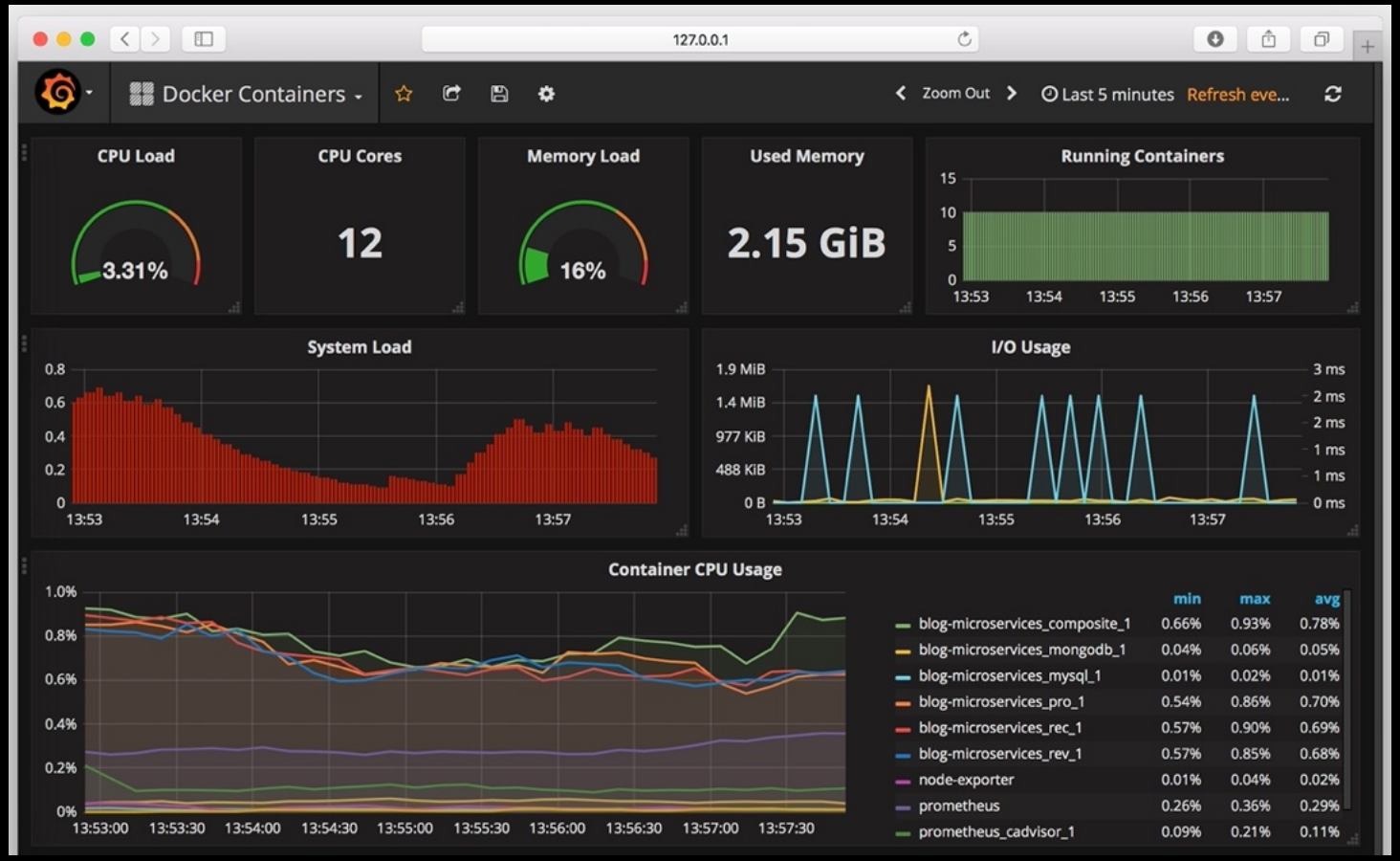


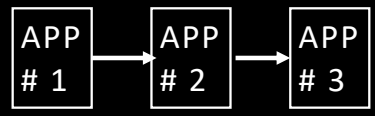


PROCESSING TIME?

OBSERVABILITY

- Observability = Logging + Tracing + **Metrics**

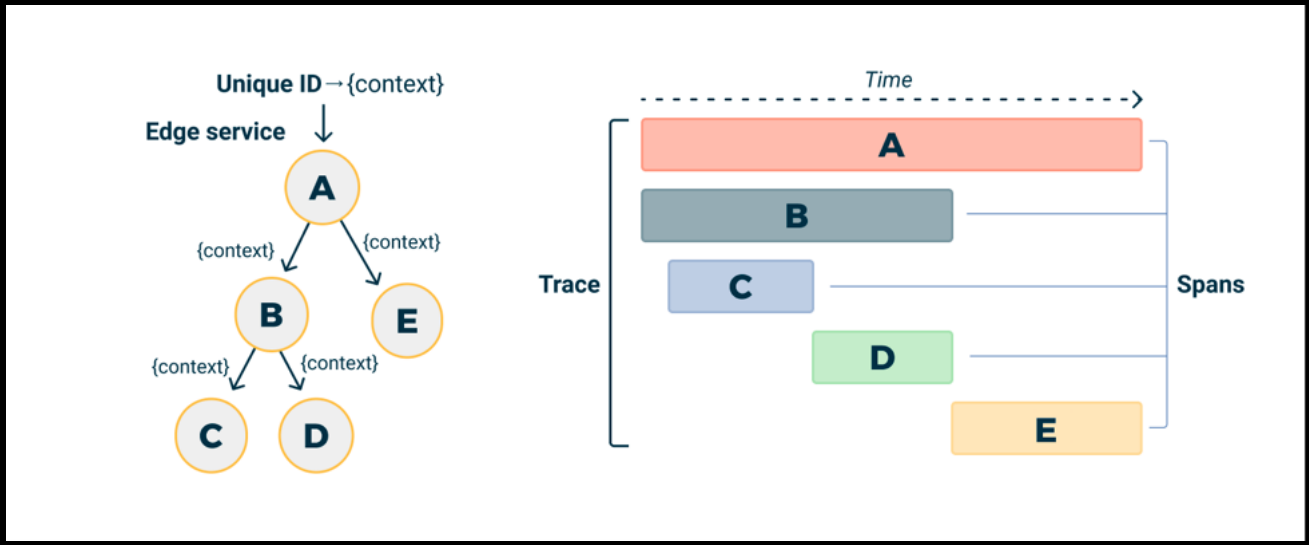




RESOURCE USAGE?

OBSERVABILITY

- Observability in Spring Framework 6.0
  - **Logs** and **Metrics** already supported in 5.0
  - **Tracing**: New module, **Micrometer Tracing**
    - » Based on **spring-cloud-sleuth**
    - » Traces are reported as a **trace tree** of **spans** based on OpenTelemetry
    - » **Contexts** based on W3C Trace Context



OBSERVABILITY

- Tracing in Spring Framework 6.0
  - Built-in support
    - » Creates traces for incoming requests, if missing
    - » Propagates to outgoing requests
    - » Supports both synchronous and asynchronous requests
    - » Propagates to logs
  - Dependencies

```
implementation 'io.micrometer:micrometer-tracing-bridge-otel'
implementation 'io.opentelemetry:opentelemetry-exporter-zipkin'
```

Bridge Observability API to OpenTelemetry

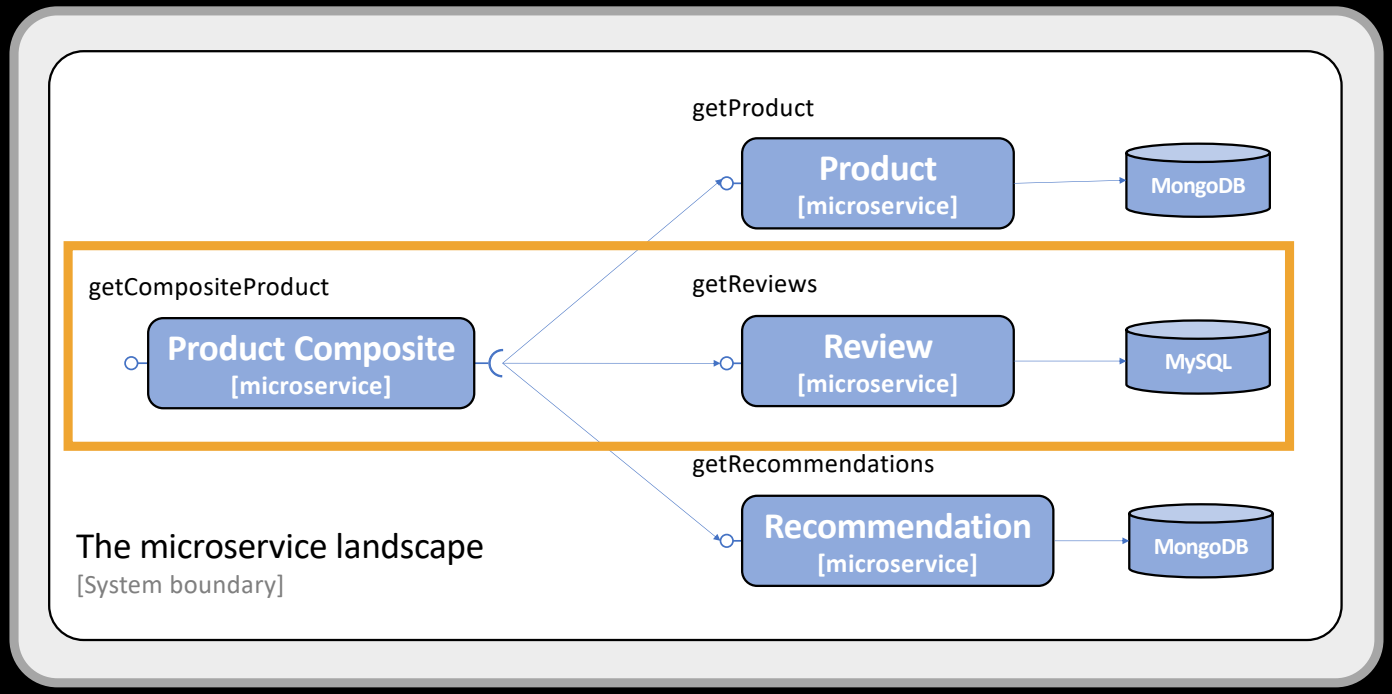
Report to a Zipkin compatible tracer

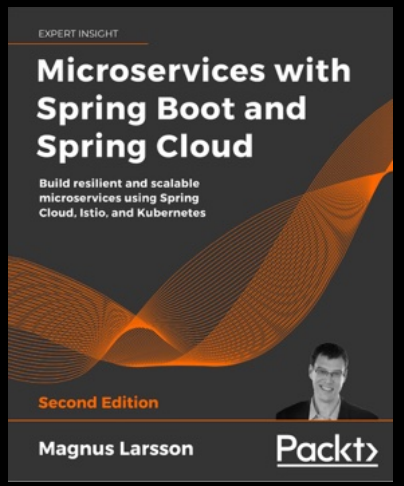
    - » Support for alternative Tracer Implementations
  - No auto propagation (yet) for reactive libraries, e.g. Spring WebFlux
    - » Spring Boot 3 Webflux project missing traceId and spanId in logs
    - » Context Propagation Library

- Tracing in Spring Framework 6.0
  - Programmatically
    - » Spring abstraction **`Observation`**
    - » Custom spans and contexts can be created

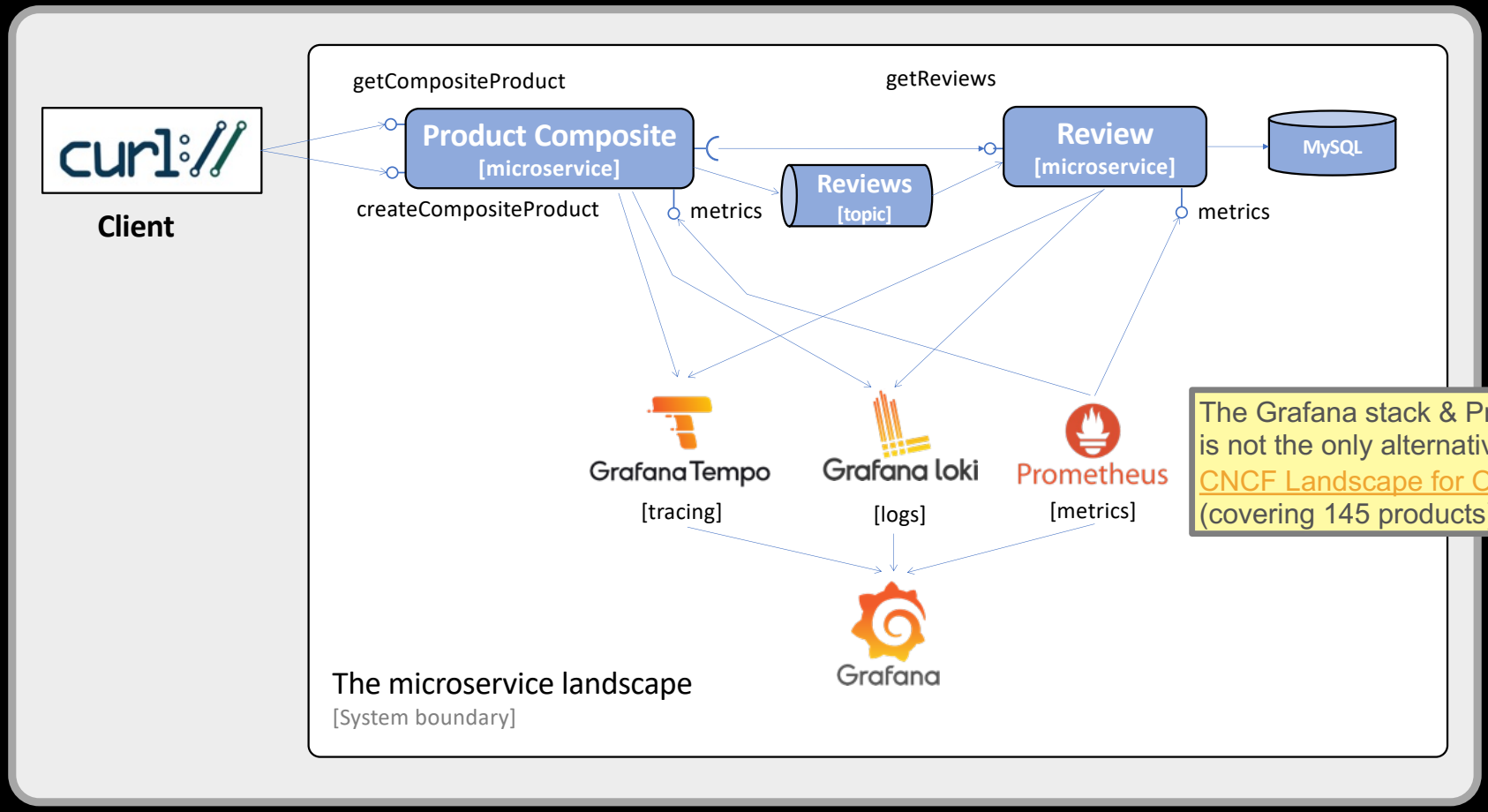
```
120     @ResponseStatus(HttpStatus.ACCEPTED)
121     @DeleteMapping(value = "/api/product-composite/{productId}")
122     void deleteProduct(@PathVariable int productId) {
123
124         Observation.createNotStarted( name: "composite.delete.observation", registry)
125             .highCardinalityKeyValue("productId", "" + productId)   <- CONTEXT KEY VALUE
126             .contextualName("composite.delete.context")             <- CONTEXT NAME
127             .observe(() -> {
128                 log.info("Will send a message for deletion of all reviews for product id {}", productId);
129                 pci.deleteReviewsBIO(productId);
130             });
131     }
```

DEMO: OBSERVABILITY

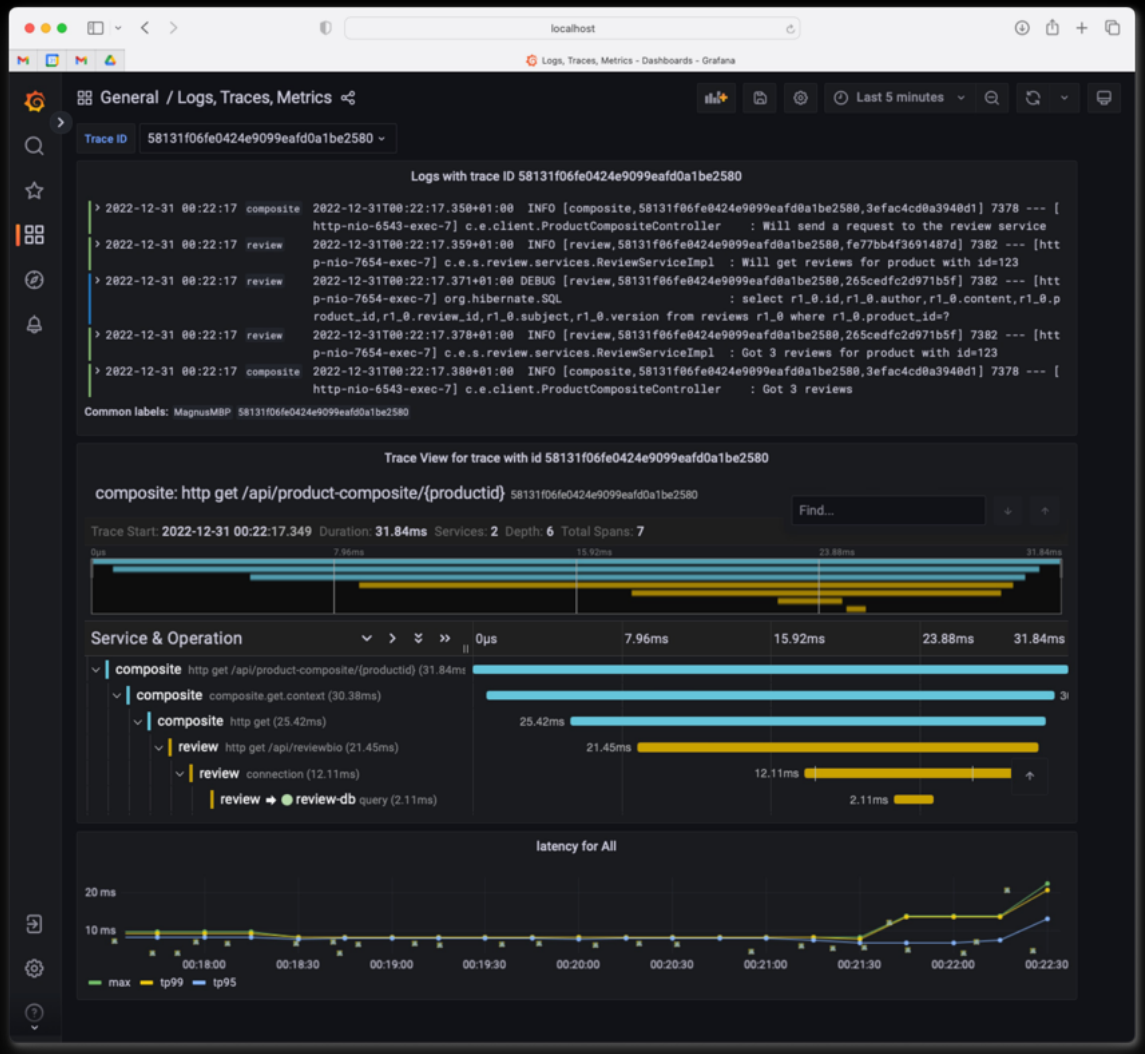




DEMO: OBSERVABILITY



DEMO: OBSERVABILITY



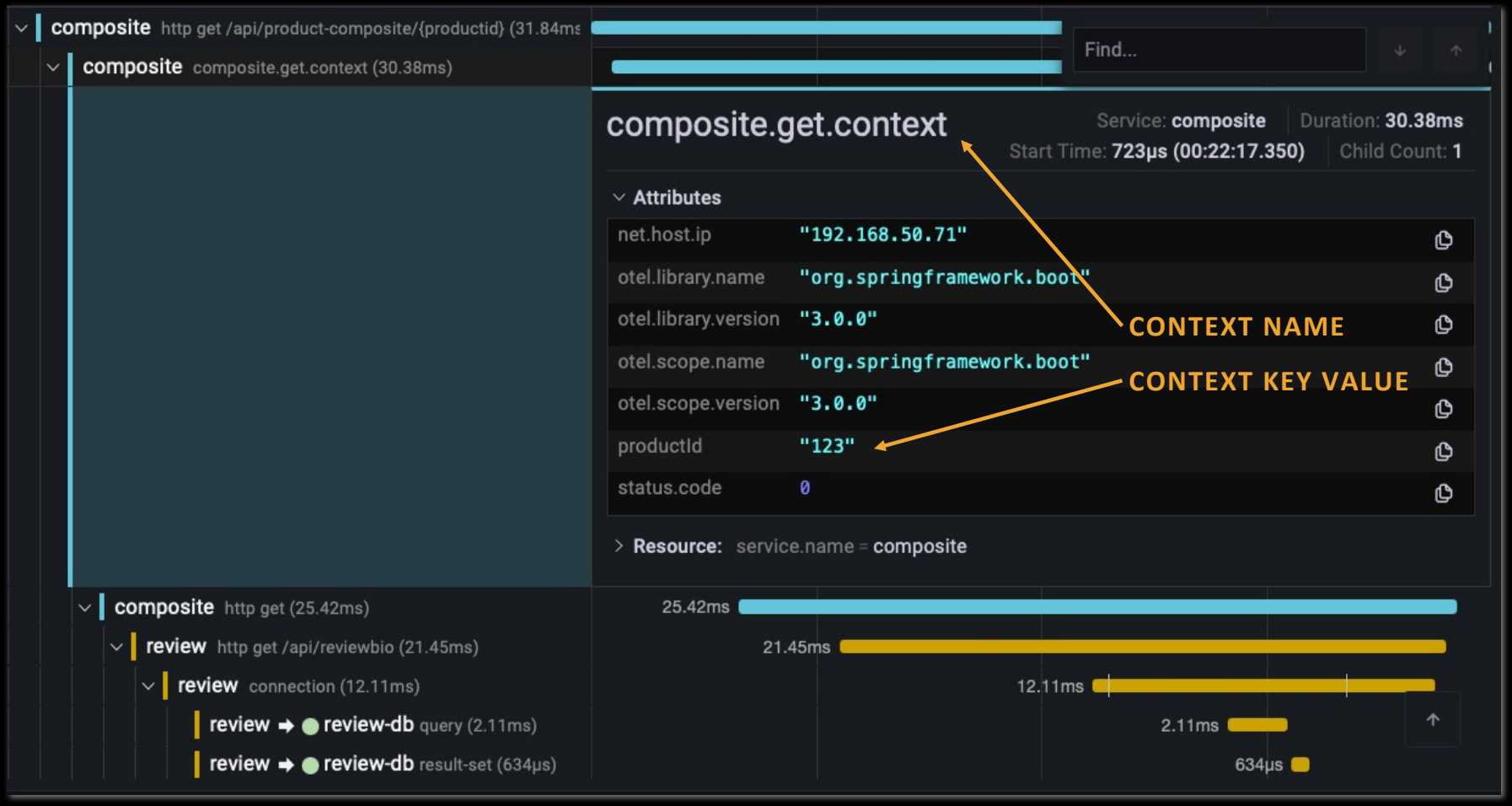
LOGS

TRACES

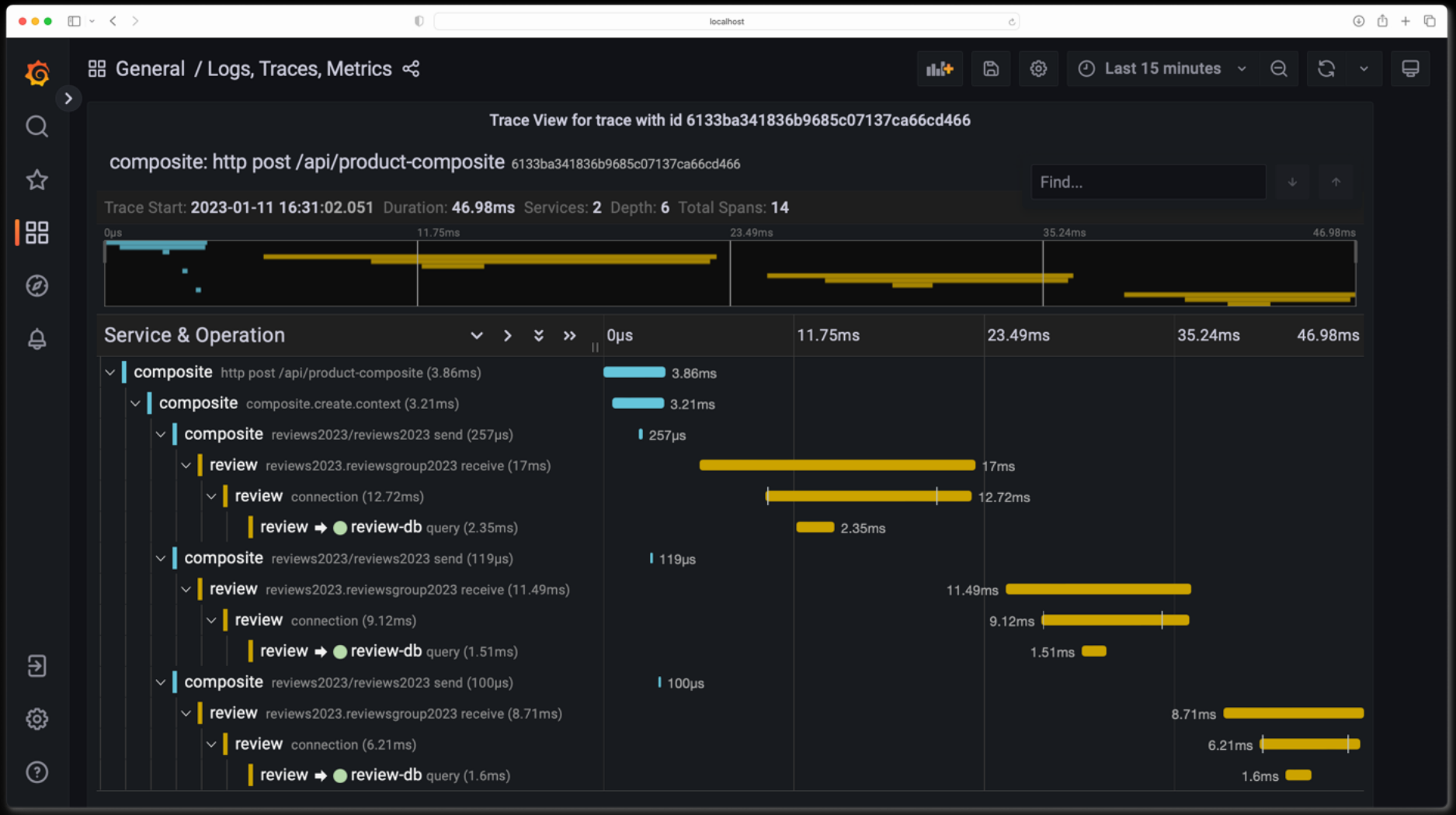
METRICS

CALLISTA

DEMO: OBSERVABILITY



DEMO: OBSERVABILITY



AGENDA

- Overview
- Migration
- Native Compile
- Observability
- **Summary**

SUMMARY

- With Spring Boot 3, a new foundation is in place
  - Expect a lot of improvements to come over the following years…
- Migration
  - Upgrade to Java 17 and `jakarta` package names
  - Remove deprecated code
- Native Compile
  - Use if start-up time is important
  - Test and build native images in CI/CD build pipeline
  - Reduce startup times running Java VM in AOT-mode
- Observability
  - Built-in auto-configuration for tracing
  - One interface, `Observation`, to abstract them all
  - One dashboard to observe them all

QUESTIONS?

ML@CALLISTAENTERPRISE.SE

MAGNUSLARSSONCALLISTA

QUESTIONS?

ML@CALLISTAENTERPRISE.SE

MAGNUSLARSSONCALLISTA